MD4 Hash Function

Team Aquinas Protocol
Ryan Schoppmeyer
Thomas Bottom
MD4 Hash Function

- Hashes inputs of arbitrary size
  - Fixed 128 bit output
- Step 1 – Padding
  - 1 followed by 0's such that the message length is congruent to 448 mod 512
  - Append a 64 bit integer representation of the length prior to the padding
  - Total length of message to hash is now a multiple of 512
MD4 Hash Function

- Step 2 – Process the message in 16 word blocks (16 words * 32 bits / word = 512 bits)
  - Three hashing functions take 3 32 bit words as input and produce a single 32 bit word as output.
  - Three rounds of hashing take place for each block, each of which make 16 hash op calls
  - State is stored in four 32 bit registers which are modified by the hash ops
MD4 Hash Function

- After all blocks have been hashed the final state is stored in the four registers (A,B,C,D)

- The hash functions output is the contents of the registers starting with the lowest order byte of A and ending with the highest order byte of D.
MD4 Hash Function - Design

- Written in C++
  - Conforms to C++ standards
  - Portable to systems with different endianness
  - Compiled and linked with GNU C++ compiler
  - Uses only C++ Standard Library
    - Primarily for I/O
  - Variable names consistent with RFC 1320
MD4 Hash Function - Design

- Original design used no preprocessor macros
  - Designed with efficiency in mind
  - Heavy computation kept to one run areas
  - Branches constructed with the most common path first
- Performance was lost due to tight loops over short functions.
MD4 Hash Function – Running Time

MD4 -t
Initializing memory block... done.
Hashing 16MB Block of random data 10 times... done. Hashed 160MB (167772160 bytes) in 2.12447 seconds (75.31264 MB/s)
MD4 Hash Function - Profile

Functions doing most individual work:

- MD4Hasher::G
- MD4Hasher::CShift
- MD4Hasher::HashOp1
- MD4Hasher::HashOp2
- MD4Hasher::HashOp3
MD4 Hash Function - Analysis

- The HashOp functions were the most frequently called operations.
  - This is because there are three rounds, each of which calls a hash operation 16 times.
  - CShift, along with the three HashOp functions, are all fairly short, each consisting of a single line. Each HashOp also contains a call to CShift.
MD4 Hash Function - Revised

- For the 16MB hash we used as a bandwidth test, there are 262,144 512 bit blocks to be hashed.
  - This means 262,144 rounds of 16 calls each to HashOp and CShift
- Implementing the HashOp and CShift functions as macros reduces the total number of function calls by 25,165,824.
MD4 Hash Function – Running Time

MD4 -t

Initializing memory block... done.
Hashing 16MB Block of random data 10 times... done. Hashed 160MB (167772160 bytes) in 1.09300 seconds (146.38609 MB/s)
MD4 Hash Function - Profile

![Bar chart showing functions doing most individual work: MD4Hasher::HashBlock, MD4Hasher::speedTest, MD4Hasher::Shuffle3, MD4Hasher::Shuffle1, MD4Hasher::Shuffle2. The chart indicates that MD4Hasher::Shuffle2 does the most individual work.]
MD4 Hash Function - Analysis

- The speedup of the optimized code versus the original running time is 1.943.

- Reducing the number of function calls by $\sim 2^{24}$ significantly reduced overhead.
MD4 Hash Function – Future Work

- There is not a lot of room to expand upon the MD4 hash function
  - Due to structural weaknesses it has been demonstrated collisions can be found in MD4 with an average of $2^8$ computations.

- Upgrades could be made to our interface (improving the use of test vectors from the command line, consistency etc.)
Optimizing code for algorithms that were designed to run quickly is hard.

- There were not a lot of ways to easily speed up the program due to the original design focus on speed.
- The use of macros for speedup exploited features of implementation rather than of design.